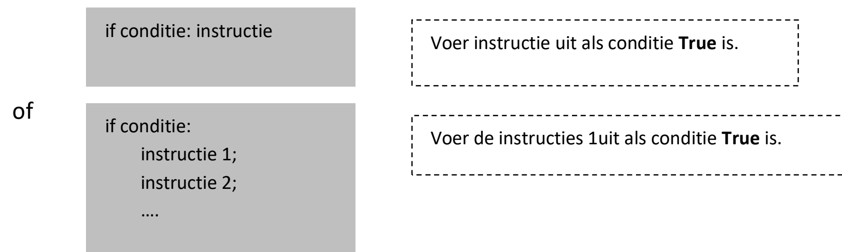
9. programmastructuren

voorwaardelijke instructies

Eén van de basiselementen van elke programmeertaal en ook van Python zijn de voorwaardelijke instructies: afhankelijk van een voorwaarde worden instructies al dan niet uitgevoerd. C heeft drie soorten: **if**, **if … else** en **if … elif**

if

De algemene vorm van een if-constructie is:



Voorbeeld:

```
a=10
if a==10:
    c = a*3
    print ("a is 10")
```

if a == 10:

Dit is de conditie, het programma gaat na of a 10 is. Indien ja, worden de 2 regels er onder uitgevoerd. Die horen bij elkaar, ze springen in.
Waarom een dubbel gelijkheidsteken? Met == vergelijken we twee waarden. if a=10: geeft een foutmelding.

if … else en if … elif

Met het else-deel geven we aan wat er moet gebeuren als de conditie niet voldaan is. Met het elif deel kan je een bijkomende voorwaarde definiëren als de eerste niet voldaan is.

Voorbeeld:

```
a=13
if a==10:
    print ("a is 10")
else:
    print("a is niet 10")
```

De conditie is niet voldaan, “a is niet 10” wordt afgedrukt

Voorbeeld:

```
a=13
if (a==10):
    print ("a is 10")
elif (a<10):
    print("a < 10")
```

De eerste conditie is niet voldaan, dan wordt de tweede getest. Die is ook niet voldaan. Er wordt niets afgedrukt.

Vergelijkingsoperatoren en logische expressies

De conditie van een if-statement is een logische uitdrukking. Meestal zie je daar een vergelijkingsoperator: ==, <, >, ≤ of ≥. Soms zal je twee of meer uitdrukkingen combineren met and, or of not. Dat is beschreven in hoofdstuk V*ariabelen in Python*.

lussen – loops

Een **lus** of **loop** is een constructie die een stuk code steeds laat herhalen. Python kent twee soorten lus: **while** en **for**.

While

While zorgt er voor dat een stuk programma herhaald wordt zolang aan een voorwaarde voldaan is.

```
while conditie:
   instructie 1;
   instructie 2;
   ….
```

De conditie wordt getest voor het uitvoeren van de loop. Zolang die conditie voldaan is, wordt het pogrammablok uitgevoerd en wordt de conditie terug getest.

Voorbeeld:

```
a=0
while a<10:
    print(a)
    a = a+1
```

We geven a beginwaarde nul. Zolang a kleiner is dan 10, wordt a afgedrukt en met 1 vermeerderd. Het resultaat? De getallen 0 tot 9 worden afgedrukt.

```
Shell
>>> %Run -c $EDITOR_CONTENT
 0
 1
 2
 3
 4
 5
 6
 7
 8
 9
>>>
```

Figuur 35: de while-lus

Soms is het handig dat een programma altijd dezelfde loop doorloopt. Dat kan met een while loop

```
while (True):
   doe iets
```

Aan de voorwaarde is altijd voldaan!

De for-loop

De for-loop heeft toch de voorkeur voor veel programmeurs omdat de volledige controle van de loop op één plaats zit. Daardoor wordt alles net iets overzichtelijker.

```
for variabele in lijst
   instructie 1;
   instructie 2;
   ….
```

Lijst is een string (lijst letters) of een lijst getallen. De instructies worden uitgevoerd voor elke waarde van de lijst. Je ziet operator *in* uit paragraaf *list* verschijnen. Alle varianten kunnen we hier ook gebruiken.

voorbeeld:

```
for letter in "RP 2040":
    print (letter)
```

De letters van de string worden onder elkaar afgedrukt.

```
Shell
>>> %Run -c $EDITOR_CONTENT
 R
 P

 2
 0
 4
 0
>>>
```

Figuur 36: for-loop

Meestal gebruiken we de for-loop voor een bereik van getallen. Range(5) zij de getallen van 0 tot 5, 5 niet-inbegrepen. Range (1, 5) zijn de gehele getallen van 1 tot 5, 5 niet-inbegrepen. In een range kunnen we een stapgrootte opgeven. Range (0, 5, 2) zijn dan de getallen 0, 2, 4.

voorbeeld:

```
for a in range (0, 5):
    print (a)
```

```
Shell
>>> %Run -c $EDITOR_CONTENT
 0
 1
 2
 3
 4
>>>
```

Figuur 37: for-loop met range

voorbeeld:

```
for a in range (0, 5 ,2):
    print (a)
```

```
Shell
>>> %Run -c $EDITOR_CONTENT
 0
 2
 4
>>>
```

Figuur 38: for-loop met stapgrootte

break en continue

We gebruiken **break** binnen for- en while-loops om die direct beëindigen.

voorbeeld:

```
for a in range (0, 5):
    if a==3: break
    print (a)
```

0, 1 en 2 worden afgebeeld. Als a gelijk is aan 3, stopt de for-loop.

Continue slaat de rest van de huidige iteratie van de loop over. We gaan verder met de volgende iteratie.

voorbeeld:

```
for a in range (0, 5):
    if a==3: break
```

```
    print (a)
```

0, 1, 2 en 4 worden afgebeeld. Als a gelijk is aan 3, stopt de iteratiestap met a=3 en gaan we verder met a=4.

Try, except, else omgaan met ongewenste gebeurtenissen

De meest voorkomende fouten in een programma zijn syntax-fouten: fouten tegen de grammaticale regels van Python. Een voorbeeld is de regel

print (“Micropython)

Bij het uitvoeren verschijnt de boodschap “SyntaxError:”, hier is een aanhalingsteken vergeten.

Soms is een programma grammaticaal correct maar loopt loopt het toch mis. We lezen een sensor uit, maar die is defect, we willen een bestand lezen maar dat bestaat niet of we delen door een variabele maar die is nul. We noemen deze soort fouten **exceptions** en we kunnen ze opvangen in het programma met een **try**: **except**: **else**: constructie.

Voorbeeld:

```
quotient = teller / noemer
print (quotient)
```

Deze code kan een probleem geven (als noemer = 0). We lossen dit op met een try:-constructie:

```
try:
    quotient = teller / noemer
except:
    print ("fout: noemer wordt nul")
else:
    print (quotient)
try:
    quotient = teller / noemer
except ZeroDivisionError:
    print ("fout: noemer wordt nul")
except TypeError:
    print ("niet-compatibele datatypes")
else:
    print (“geen fouten”)
teller = 10.0
noemer = 'ee'
teller = 10.0
noemer = 0
```